Class06: R Functions

Isabella Ruud

Today we are going to get more exposure to functions in R.

Let's start with a silly simple function to add some numbers:

```
#add two numbers together with a default value of 0 for y
add <- function(x,y=0) {
  x + y
}
```

Can we use this function? Yes, if you play the code block with the function defined in it

```
add(1,1)
```

```
[1] 2
```

```
add(c(100,200),1)
```

```
[1] 101 201
```

```
add(x=100, y=1)
```

```
[1] 101
```

Sometimes you need to argue with functions to get it to do what you want (ie define what base to calculate the log in)

```
log(10)
```

```
[1] 2.302585
```

```
log(10,base=10)
```

```
[1] 1
```

You need to pass the right number of arguments to the function for it to run

```
#this code brings up an error message so I commented it out
#add(100,1,200)
```

sample() function takes a sample of a specified size from the elements of x with or without replacement AKA it randomly selects elements from a vector

It takes 4 arguments x: a vector of elements to choose from, n: the number of items to choose, replace: defaults to false and tells whether to sample with replacement, and prob: degaults to null and can specify probability weights for choosing elements from the vector

```
sample(1:10, size = 1)
```

```
[1] 2
```

```
sample(1:10, size=11, replace = TRUE)
```

```
 [1]  7  5  4  7  9  9  6  2  1  3 10
```

Note that you can only draw a sample larger than the vector size if replace is set to true.

Generate DNA sequences

Write a function to generate a random nucleotide sequence of a use specified size

All functions in R have at least 3 things: -a name (we pick this, in this case it is generate_DNA) -input arguments (length of the output sequence) -body (randomly samples from A,T,G,C characters and pastes the characters into a string)

```
#generate_DNA(length) takes an input length and makes a random DNA sequence of that length
generate_DNA <- function(length) {

#take length number of random samples from the list of nucleotides with replacement
nuc_list <- sample(c('A', 'T', 'G', 'C'), size = length, replace = TRUE)

#print out a single element vector by concatenating the characters from the random sampling t
dna <- paste(nuc_list, collapse = "")
return(dna)
}

generate_DNA(6)
```

```
[1] "GACGGG"
```

I want the ability to switch between two output formats (concatenating the nucleotides or not), so I can do that with an extra input argument that controls it with TRUE/FALSE

```
#generate_DNA(length) takes an input length and makes a random DNA sequence of that length. r
generate_DNA <- function(length, collapse = TRUE) {

#take length number of random samples from the list of nucleotides with replacement
dna <- sample(c('A', 'T', 'G', 'C'), size = length, replace = TRUE)

#print out a single element vector by concatenating the characters from the random sampling t
if(collapse) {
  dna <- paste(dna, collapse = "")
}
return(dna)
}

generate_DNA(6)
```

```
[1] "GAGGAC"
```

```
generate_DNA(6, collapse=FALSE)
```

```
[1] "A" "A" "A" "C" "A" "G"
```

Add the ability to print out a message if the user is sad using an input parameter called mood where if mood is false, then the user is sad and the message is printed out

```
#generate_DNA(length) takes an input length and makes a random DNA sequence of that length. r

generate_DNA <- function(length, collapse = TRUE, mood=TRUE) {

#take length number of random samples from the list of nucleotides with replacement
dna <- sample(c('A', 'T', 'G', 'C'), size = length, replace = TRUE)

#print out a single element vector by concatenating the characters from the random sampling t
if(collapse) {
  dna <- paste(dna, collapse = "")
}

#print out a message that the user is sad is the mood parameter is set to false
if(mood == FALSE){
  cat("user is sad :(\n")
}
return(dna)
}

generate_DNA(6)
```

```
[1] "CGCTGA"
```

```
generate_DNA(6, mood = FALSE)
```

```
user is sad :(
```

```
[1] "GTAGAT"
```

Generate a protein of any length

```
#generate_protein() takes an input length and then samples that number of amino acids from a
generate_protein <- function(length, collapse = FALSE) {
  amino_acids <- c("A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R"
  protein <- sample(amino_acids, length, replace=TRUE)
  if(collapse){
    protein <- paste(protein, collapse="")
```

```
  }
  return(protein)
}

#test out the function
generate_protein(20, collapse = TRUE)
```

```
[1] "YMHPDMSPKGRMGTDVSVLY"
```

Generate protein sequences from length 6 to 12 amino acids long

```
#make a vector of lengths from 6 to 12 amino acids long
prot_lens <- c(6:12)

#use sapply to apply the generate_protein function to the prot_lens list. sapply will apply t
myseqs <- sapply(prot_lens, generate_protein, collapse=TRUE)
myseqs
```

```
[1] "WKRQMS"       "HIEMAGA"      "SLTKTTWL"     "ESWIPCSLK"    "HMQHCCKRWW"
[6] "GAYPTWVVWAW"  "RWVGGTHGCKVS"
```

Are any of the sequences unique in the sense that they have never been found in nature?

To make this accessible, lets get our sequences in FASTA format. FASTA format looks like this

id.6 MTRFDEEASWYT id.7 MHTYPMNNG

The functions paste() and cat() will help here

```
cat(paste(">id.",6:12, "\n" , myseqs, "\n", sep = ""), sep="")
```

```
>id.6
WKRQMS
>id.7
HIEMAGA
>id.8
SLTKTTWL
>id.9
ESWIPCSLK
>id.10
```

```
HMQHCCKRWW
>id.11
GAYPTWVVWAW
>id.12
RWVGGTHGCKVS
```

When I blast these sequences, at sequence length = 9, the percent identity and coverage are not both 100%, indicating that this sequence is not seen in nature

```
#install.packages("bio3d")
#library(bio3d)
#myseqs <- sapply(prot_lens, generate_protein, collapse=TRUE)
#as.matrix(myseqs)
#x <- as.fasta(as.matrix(myseqs))
#x
```

> Q1. Write a function grade() to determine an overall grade from a vector of student homework assignment scores dropping the lowest single score. If a student misses a homework (i.e. has an NA value) this can be used as a score to be potentially dropped. Your final function should be adquately explained with code comments and be able to work on an example class gradebook such as this one in CSV format: "https://tinyurl.com/gradeinput"

```
#grade is a function that takes a list of grades that contain numbers and NAs for missed ass

grade <- function(grades) {

  #grades_sorted is a list of the grades sorting from the highest grade to the lowest grade
  grades_sorted <- sort(grades, decreasing = TRUE, na.last = TRUE)

  #grades_dropped removes the last grade in grades_sorted to drop the lowest grade, which wi
  grades_dropped <- grades_sorted[1:length(grades_sorted)-1]

  #the average of the grades after the lowest score is removed is calculated. First the sum
  return(sum(grades_dropped, na.rm = TRUE) / length(grades_dropped))
}

#testing the grade function with vectors that have 0, 1, or multiple NAs and making sure the
grade(c(98, 75, 43, 50))
```

```
[1] 74.33333
```

```r
grade(c(87, NA, 54, 100, 78))
```

```
[1] 79.75
```

```r
grade(c(87, 0, 54, 100, 78))
```

```
[1] 79.75
```

```r
grade(c(100, NA, 99, 102, NA, 34))
```

```
[1] 67
```

```r
grade(c(100, 0, 99, 102, 0, 34))
```

```
[1] 67
```

```r
#making sure the grade function can work on vectors from the gradebook csv
gradebook <- read.csv("https://tinyurl.com/gradeinput")

#the grade_csv has rows of students and columns for each homework assignment. I ran head(grad
#head(grade_csv)

#you have to convert the row from the dataframe to a numeric vector before passing it to grad
grade(as.numeric(as.vector(gradebook[1, -1])))
```

```
[1] 91.75
```

Q2. Using your grade() function and the supplied gradebook, Who is the top scoring student overall in the gradebook?

student-18 is the top scoring student overall with an overall score of 94.5

```r
#load in the gradebook from the link as a dataframe
gradebook <- read.csv("https://tinyurl.com/gradeinput")

#make a column in the gradebook dataframe that contains the overall grade calculated by the g
gradebook$overall_grade <- apply(gradebook[,-1], 1, function(x) grade(as.numeric(as.vector(x)
```

```
#sort the gradebook by the overall_grade column. I set the decreasing to true so that the to
sorted_gradebook <- gradebook[order(gradebook$overall_grade, decreasing = TRUE),]

#print out the first row of the sorted gradebook, which will contain the student with the to
sorted_gradebook[1,]
```

```
            X hw1 hw2 hw3 hw4 hw5 overall_grade
18 student-18  91  NA 100  87 100          94.5
```

Q3. From your analysis of the gradebook, which homework was toughest on students (i.e. obtained the lowest scores overall?

```
avg_nas <- function(scores){
  sum <- sum(scores, na.rm = TRUE)
  length <- length(scores)
  return(sum/length)
}
gradebook.t <- t(gradebook)
gradebook.t$hw_avg <- apply(gradebook.t, 1, function(x) avg_nas(as.numeric(as.vector(x))))
```

```
Warning in avg_nas(as.numeric(as.vector(x))): NAs introduced by coercion
```

```
Warning in gradebook.t$hw_avg <- apply(gradebook.t, 1, function(x)
avg_nas(as.numeric(as.vector(x)))): Coercing LHS to a list
```

```
gradebook.t$hw_avg
```

```
           X           hw1           hw2           hw3           hw4
       0.000        89.000        72.800        80.800        85.150
         hw5 overall_grade
      79.250        87.425
```